Logika

МОДУЛЬ 4. УРОК 1

Урок 1. Django Intro

Модуль 4. Урок 1

Django Structure

Django

Django - це високорівневий Python веб-фреймворк, що заохочує швидку розробку та чистий, прагматичний дизайн. Використовується для створення масштабованих та забезпечених веб-додатків.

https://docs.djangoproject.com/en/5.0/

Django

Для встановлення Django, використовуйте pip.

Команда: pip install django.

Після встановлення, можна створити новий проект за допомогою:

django-admin startproject myproject

MVT (Model-view-template)

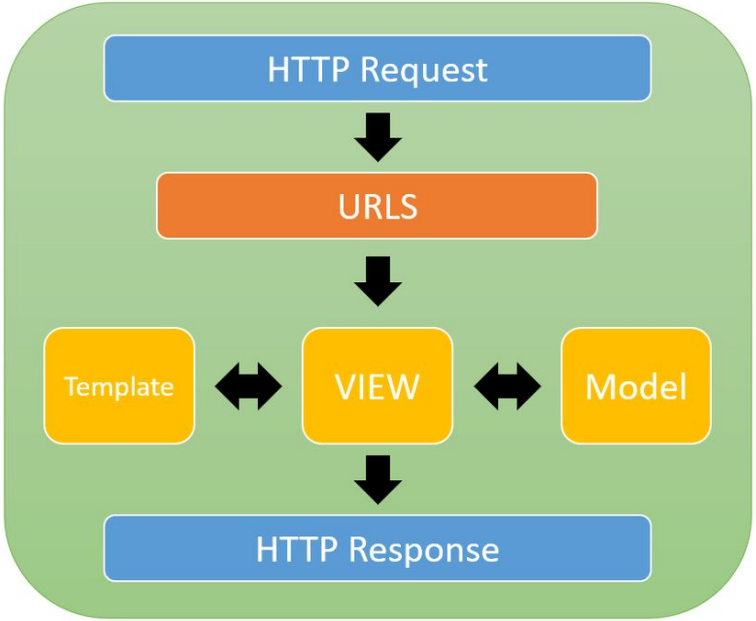
Модель: Також відповідає за структуру даних і взаємодію з базою даних, подібно до MVC.

View: У Django, View виконує роль контролера в традиційному **MVC**. Він обробляє бізнес-логіку: отримує вхідні дані від користувача, взаємодіє з моделлю, і передає дані до шаблону.

Template: Виконує роль View в MVC. Це шар, що відповідає за відображення даних, які були оброблені і передані View. Шаблони визначають, як дані будуть відображені на сторінці.

Структура роботи Django

- Користувач відправляє запит через веб-браузер.
- Запит обробляється URL маршрутизатором, який визначає, яке представлення викликати.
- Представлення взаємодіє з моделлю за потреби та обирає відповідний шаблон.
- Шаблон рендериться з необхідними даними та відправляється користувачу як відповідь.



Django Important Files

manage.py: Утиліта командного рядка, що дозволяє виконувати багато задач, пов'язаних з проектом.
settings.py: Налаштування проекту Django, включаючи конфігурацію бази даних, статичні файли, мовні налаштування тощо.
urls.py: Файл для оголошення URL маршрутів проекту, вказує Django, які представлення використовувати для обробки різних HTTP запитів.
models.py: Визначення моделей, які Django використовує для створення схеми бази даних.
views.py: Визначення представлень, які обробляють бізнес-логіку та взаємодію з моделями для передачі даних до шаблонів.

Модуль 4. Урок 1

Django Models, Views, Templates

View (Представлення)

View у Django — це Python функція або клас, який приймає веб-запит і повертає веб-відповідь. Представлення відповідає за обробку логіки, необхідної для виконання конкретного запиту, і зазвичай взаємодіє з моделями для отримання, зберігання або зміни даних, а також з шаблонами для рендеринга відповіді у вигляді HTML (або іншого формату).

View може виконувати різноманітні задачі, включаючи валідацію форм, виконання запитів до бази даних через моделі, генерацію JSON для AJAX запитів і багато іншого.

View Example

```
from django.http import HttpResponse

def hello_world(request):
 return HttpResponse("Hello, world!")
```

Цей **view** повертає **HTTP** відповідь з текстом "Hello, world!".

View в **Django** приймають **HTTP** запит як параметр та повертають **HTTP** відповідь.

Templates (Шаблони)

Templates у Django — це файли, які використовуються для генерації відповідей користувачам, переважно у вигляді HTML. Шаблони містять статичний **HTML** код та спеціальні позначки **Django Template Language (DTL)**, які дозволяють динамічно вставляти контент і використовувати логіку програмування, таку як цикли та умовні оператори, безпосередньо у **HTML**.

Шаблони дозволяють вам відділити дизайн веб-сторінки від Python коду, роблячи ваш код більш чистим і легшим для розуміння. Вони забезпечують гнучкість у відображенні даних, що передаються з **view**, і дозволяють легко змінювати зовнішній вигляд веб-сторінок без необхідності змінювати логіку обробки запитів.

Templates (Шаблони)

Основні можливості шаблонів включають:

Передача змінних: Використовуйте контекст, переданий з вью, для динамічного відображення даних.

Теги та фільтри: Спеціальні позначки для виконання логіки програмування прямо у шаблоні (наприклад, **{% for post in posts %}**) та фільтри для форматування відображення даних (**{{ post.date|date:"D d M Y" }}**).

Спадкування шаблонів: Дозволяє створювати базові "скелети" веб-сторінок, які інші шаблони можуть розширювати, що спрощує управління зовнішнім виглядом веб-додатку.

Найуживаніші template tags

{% for %} і {% endfor %} для ітерації по списку.

{% if %} і {% endif %} для умовних виразів.

{% url 'viewname' %} для генерації URL до вьюшки.

URLS (шляхи)

```
from django.urls import path
from . import views

urlpatterns = [
    path('hello/', views.hello_world),
]
```

В цьому прикладі, запит до **/hello/** буде оброблено **view hello_world**.
urls.py визначає маршрутизацію **HTTP** запитів до відповідних **view**

Основні команди Django

django-admin startproject projectname: Створення нового проекту.
python manage.py startapp appname: Створення нового додатку в проекті.
python manage.py makemigrations: Створення нових міграцій на основі змін у моделях.
python manage.py migrate: Застосування міграцій до бази даних.
python manage.py runserver: Запуск розробницького сервера.
python manage.py createsuperuser: Створення адміністратора для доступу до адмін-панелі Django.

Logika

МОДУЛЬ 4. УРОК 2

Урок 2. Моделі та база даних

Модуль 4. Урок 2

Django models, Django ORM

Визначення ORM та його роль у Django:

ORM (Object-Relational Mapping) дозволяє взаємодіяти з базою даних за допомогою Python об'єктів, замість використання SQL. Це ключовий компонент Django, який спрощує розробку та підтримку додатків.

Переваги використання ORM для роботи з базою даних:

- Абстракція коду та незалежність від СУБД.
- Зниження ризику SQL ін'єкцій.
- Спрощення процесу розробки завдяки використанню Python синтаксису.

Основні методи Django ORM:

- **.all**(): Повертає всі записи з таблиці.
- **.filter**(): Фільтрує записи за заданими критеріями.
- **.exclude**(): Виключає записи, які відповідають заданим критеріям.
- **.get**(): Повертає один запис, який точно відповідає критеріям.
- **.create**(): Створює новий запис у таблиці.
- **.update**(): Оновлює вибрані записи.
- **.delete**(): Видаляє вибрані записи.

Django Models

Модель у Django - це Python клас, який відображається на таблицю у базі даних. Моделі визначають структуру даних, їх властивості та поведінку.

Django Models

CharField для коротких текстових рядків.
TextField для довгих текстових рядків без обмежень по довжині.
IntegerField для цілих чисел.
DateField та DateTimeField для дат та дат-часів відповідно.
EmailField спеціалізоване для електронних адрес.
BooleanField для істинності/хибності (True/False).
FileField та **ImageField** для зберігання файлів та зображень.
DecimalField для чисел з фіксованою кількістю десяткових знаків, корисно для фінансових даних.

Зв'язки між моделями

- One-to-One (OneToOneField): Зв'язок один до одного, наприклад, користувач та його профіль.
- ForeignKey (One-to-Many): Зв'язок один до багатьох, наприклад, користувач та його замовлення.
- ManyToManyField: Множинний зв'язок, наприклад, книги та їх автори.

class Meta

Налаштування моделей через class Meta:

Клас Meta використовується в моделях Django для визначення додаткових параметрів, як-от:

- Сортування за замовчуванням (ordering)
- Назви таблиць (db_table)
- Вербальні назви моделей (verbose_name і verbose_name_plural)

Logika

МОДУЛЬ 4. УРОК 3

Урок 3. Представлення, шаблони та маршрутизація

Модуль 4. Урок 3

Django Views

Django Views

В'ю (View) у Django - це функція Python або клас, що приймає веб-запит і повертає веб-відповідь. В'ю визначає логіку обробки запитів користувача. Воно може генерувати HTML відповідь, перенаправляти на іншу сторінку, обробляти форми тощо.

Django Views

Клас-бейсд в'ю (Class-Based Views, CBVs) у Django - це спосіб організації логіки в'ю за допомогою класів.

Вони надають більше можливостей для повторного використання коду та спрощення розширення функціоналу порівняно з функціональними в'ю

Django Views

CBV EXAMPLE:

```
from django.views.generic import ListView
from .models import Book

class BookListView(ListView):
    model = Book
    template name = 'books/book list.html'
```

Приклад використання клас-бейсд в'ю для створення сторінки зі списком книг. ListView автоматично обробляє більшість задач, пов'язаних з відображенням списку об'єктів

Django Views

В'ю у вигляді функцій (Function-Based Views, FBVs) - це спосіб організації в'ю за допомогою звичайних функцій. Вони дозволяють легко читати та розуміти логіку обробки кожного запиту, якщо логіка не надто складна.

Django Views

FBV EXAMPLE:

```
from django.shortcuts import render

def book_list(request):
    books = Book.objects.all()
    return render(request, 'books/book_list.html',
{'books': books})
```

Простий приклад функціонального в'ю для відображення списку книг. Функція приймає запит та повертає HTML-відповідь з використанням шаблону.

Django Views

Class-based в'ю рекомендуються для складних інтерфейсів з багатьма спільними елементами, тоді як **function-based в'ю** ідеально підходять для простіших сценаріїв з чіткою логікою. Вибір залежить від ваших потреб та переваг у структуризації коду.

Також, варто звертати увагу на кастомізацію обробки запитів.

Модуль 4. Урок 3

URLS (маршрутизація)

Маршрутизація

Маршрутизація у Django - це процес визначення URL-шляху, який відповідає конкретному в'ю. Вона дозволяє зв'язати URL-адресу з функцією або класом в'ю, що обробляє запит.

Маршрутизація

Цей приклад демонструє, як у файлі **urls.py** можна налаштувати маршрути для **CBV** та **FBV**, використовуючи **path** для визначення шляху **URL** і вказівки на відповідне в'ю.

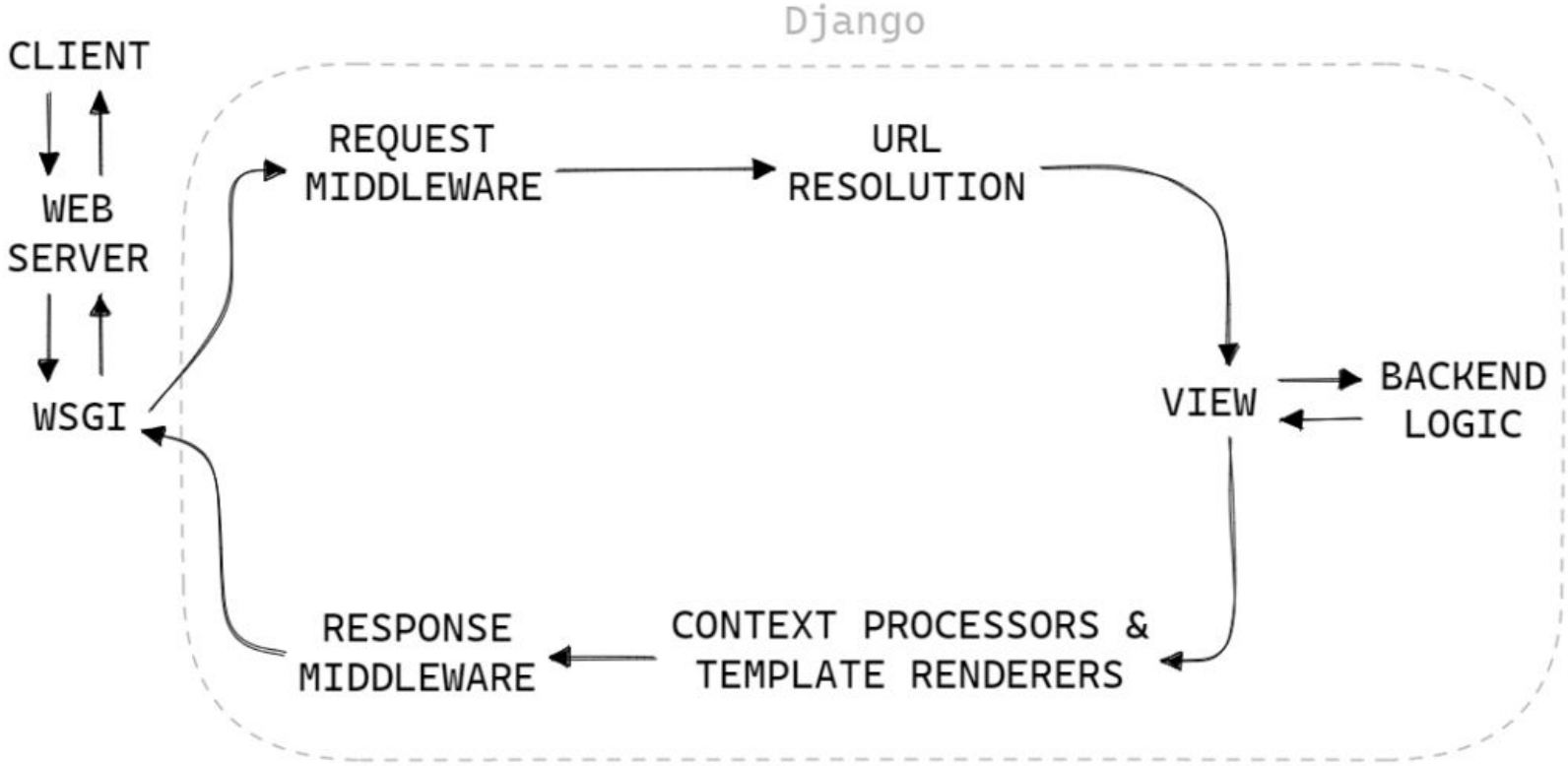
```
from django.urls import path
from .views import BookListView, book list

urlpatterns = [
    path('books/', BookListView.as view(),
name='book-list-cbv'),
    path('books_fbv/', book list, name='book-list-fbv'),
]
```

Модуль 4. Урок 3

Обробка запитів та відповідей

Як обробляються запити?



Як обробляються запити?

1. **Запит (Request)** користувача до веб-сервера.
2. **URLresolver** перевіряє URL запиту проти URLconf (urls.py) для визначення відповідного в'ю.
3. **В'ю обробляє запит**. Якщо потрібно, звертається до моделі для отримання або зберігання даних у базі даних.
4. В'ю може використовувати шаблон для створення HTML відповіді.
5. **В'ю** повертає відповідь до **URLresolver**.
6. **Відповідь (Response)** передається користувачеві.

Як обробляються відповіді?

1. **В'ю** генерує відповідь: може включати обробку даних моделі, використання шаблонів для генерації HTML, або пряме формування відповіді.
2. **Проміжне ПЗ (middleware)** може модифікувати або перевіряти відповідь.
3. **Відповідь** передається веб-серверу.
4. Веб-сервер відправляє відповідь користувачу

Logika

МОДУЛЬ 4. УРОК 4

Урок 4. HTML, CSS, JS in Django

Static files

Django обробляє **статичні файли** через механізм, що дозволяє легко інтегрувати **CSS, JavaScript**, зображення та інші ресурси у веб-проекти. Використовуючи налаштування **STATIC_URL** та **STATIC_ROOT**, Django визначає, де зберігати та як отримувати статичні файли. У режимі розробки (**DEBUG=True**) Django автоматично обслуговує статичні файли за допомогою вбудованого сервера. Для продакшну, рекомендується налаштувати веб-сервер (наприклад, **Nginx або Apache**) для обслуговування цих файлів безпосередньо з **STATIC_ROOT**, забезпечуючи оптимальну продуктивність.

Static files

Для роботи зі статичними файлами в Django необхідно налаштувати кілька ключових параметрів у файлі settings.py вашого проекту.
Перший крок - визначення **STATIC_URL**, який є URL-адресою, за якою будуть доступні статичні файли. Наприклад, якщо ви встановите **STATIC_URL = '/static/'**, всі ваші статичні файли будуть доступні за адресою **http://example.com/static/.**

Static files

Далі, вам може знадобитися встановити **STATICFILES_DIRS**, список директорій, де **Django** буде шукати статичні файли, які не знаходяться внутрішньо в додатках. Це дозволяє централізовано управляти статичними файлами, які використовуються на різних сторінках вашого сайту.

Модуль 4. Урок 4

HTML, CSS

HTML tags

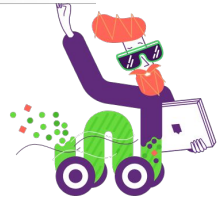
Основні теги HTML включають:
<html> для визначення кореня документа,
<head> для метаданих,
<title> для заголовка сторінки,
<body> для змісту.

HTML tags

<h1>-<h6> використовуються для заголовків,
<p> для абзаців,
<a> для посилань,
<img> для зображень,
** і ** для ненумерованих та нумерованих списків відповідно,
<li> для елементів списку,
** ** для розділів контенту,
**** для інлайнового вмісту.

Connect CSS to HTML

```
<link rel="stylesheet" href="some_file.css">
```



CSS general parameters

color: Визначає колір тексту.
background-color: Встановлює колір фону елемента.
font-family: Визначає шрифт тексту.
font-size: Встановлює розмір шрифту.
margin: Визначає зовнішні відступи навколо елемента.
padding: Встановлює внутрішні відступи елемента.
border: Додає рамку навколо елемента.
width і height: Визначають ширину та висоту елемента.
display: Керує способом відображення елемента (наприклад, block, inline, flex).
position: Визначає спосіб позиціонування елементів.

Bootstrap

https://getbootstrap.com/docs/5.3/getting-started/introduction/

Bootstrap — це популярний фреймворк фронтенд-розробки, який дозволяє швидко створювати адаптивні веб-сайти. Він містить набір інструментів для стилізації HTML-елементів, включаючи шаблони макету, компоненти для навігації, форми та кнопки, а також JavaScript плагіни.
Bootstrap пропонує систему сітки, що спрощує розробку відгуків на різних пристроях, забезпечуючи консистентність дизайну і використання.

Logika

МОДУЛЬ 4. УРОК 5

Урок 5. Django Authentication System

Authentication VS Authorization

Автентифікація — це процес перевірки особистості користувача, наприклад, через логін і пароль, для встановлення його особи в системі.

Авторизація — це наступний крок після автентифікації, який визначає права та доступи користувача до ресурсів та функцій системи.

Модуль 4. Урок 5

Django Authentication components

Django Auth Components

Основні компоненти включають:

- модель User для зберігання інформації про користувачів,
- механізми для аутентифікації та авторизації користувачів,
- систему дозволів для визначення прав користувачів,
- а також використання сесій та cookies для підтримки стану аутентифікації між запитами.

Всі ці елементи інтегруються безпосередньо у фреймворк Django

Django Auth User

`User` model ¶

class **models.User** ¶

Fields ¶

class **models.User**

User objects have the following fields:

username ¶

Required. 150 characters or fewer. Usernames may contain alphanumeric, _, @, +, . and - characters.

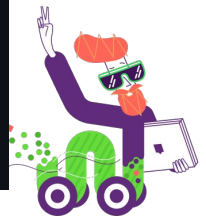
The **max_length** should be sufficient for many use cases. If you need a longer length, please use a custom user model. If you use MySQL with the **utf8mb4** encoding (recommended for proper Unicode support), specify at most **max_length=191** because MySQL can only create unique indexes with 191 characters in that case by default.

first_name ¶

Optional (**blank=True**). 150 characters or fewer.

last_name ¶

Optional (**blank=True**). 150 characters or fewer.



Django Auth User

email ¶

Optional (**blank=True**). Email address.

password ¶

Required. A hash of, and metadata about, the password. (Django doesn't store the raw password.) Raw passwords can be arbitrarily long and can contain any character. See the password documentation.

groups ¶

Many-to-many relationship to **Group**

user_permissions ¶

Many-to-many relationship to **Permission**

is_staff ¶

Boolean. Allows this user to access the admin site.

is_active ¶

Boolean. Marks this user account as active. We recommend that you set this flag to **False** instead of deleting accounts. That way, if your applications have any foreign keys to users, the foreign keys won't break.

This doesn't necessarily control whether or not the user can log in. Authentication backends aren't required to check for the **is_active** flag but the default backend (**ModelBackend**) and the **RemoteUserBackend** do. You can use **AllowAllUsersModelBackend** or **AllowAllUsersRemoteUserBackend** if you want to allow inactive users to login. In this case, you'll also want to customize the **AuthenticationForm** used by the **LoginView** as it rejects inactive users. Be aware that the permission-checking methods such as **has_perm()** and the authentication in the Django admin all return **False** for inactive users.

is_superuser ¶

Boolean. Treats this user as having all permissions without assigning any permission to it in particular.

last_login ¶

A datetime of the user's last login.

Django Auth User

Methods ¶

class **models.User**

get_username() ¶

Returns the username for the user. Since the **User** model can be swapped out, you should use this method instead of referencing the username attribute directly.

get_full_name() ¶

Returns the **first_name** plus the **last_name**, with a space in between.

get_short_name() ¶

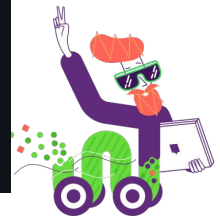
Returns the **first_name**.

set_password(*raw_password*) ¶

Sets the user's password to the given raw string, taking care of the password hashing. Doesn't save the **User** object.

When the **raw_password** is **None**, the password will be set to an unusable password, as if **set_unusable_password()** were used.

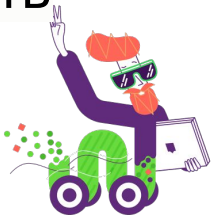
check_password(*raw_password*) ¶



How Django uses sessions?

Сесії в Django використовуються для зберігання інформації про стан користувача на сервері між різними запитами. Кожна сесія асоційована з унікальним ідентифікатором (session key), який зберігається у cookie на стороні клієнта.

Це дозволяє Django ідентифікувати запити від одного і того ж користувача та надавати відповідний контекст або дані, такі як дані про автентифікацію користувача. Django забезпечує гнучке управління сесіями, дозволяючи зберігати сесії у різних місцях, наприклад, у базі даних, на файловій системі, у кеші або навіть використовуючи зовнішні сховища.



Модуль 4. Урок 5

Django Forms

Django Forms

Форми в Django — це потужний інструмент для створення та обробки даних форм на веб-сторінках. Вони дозволяють легко збирати, валідувати та обробляти введені користувачем дані, забезпечуючи безпеку та ефективність. Django надає два типи форм:

- **Form** для загальних форм
- **ModelForm** для форм, що прямо зв'язані з моделями даних.

Django Forms (Example)

```python
from django import forms

class ContactForm(forms.Form):
    name = forms.CharField(label='Ім’я', max_length=100)
    message = forms.CharField(widget=forms.Textarea,
label='Повідомлення')
```

Django Auth Forms

AuthenticationForm - Автентифікація користувача (визначення того, який об’єкт користувача зараз належить тому, хто зробив запит)

UserCreationForm - Форма для здійснення реєстрації користувача (створення об’єкту User)